## Reflection

## 1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consists of the following steps:
1. Convert the image to grayscale
2. Apply Gaussian smoothing to the grayscale image
3. Detect edges using the Canny algorithm
4. Region masking
5. Detect line segments using Hough transform
6. Intrapolate between the line segments - create continous intrapolated left and right line
7. Extrapolate the left and right line end points from the bottom of the picture to the top of the polygon
8. Draw the extrapolated lines on the copy of the original image

In order to draw a single line on the left and right lanes, I modified the draw_lines() function (in my pipeline called "process_test_image()") by (a) classifying the line segments into "left" and "right" line based on the slope, (b) interpolating between the left segments and right segments to create continous lines and (c) finally extrapolating the continous lines to the bottom of the image and towards the top of the masked polygon (both intrapolation and extrapolation are based on the pythagorean theorem.

## 2. Identify potential shortcomings with your current pipeline

a) The pipeline creates straight lines, i.e. it won't work well with curves - partly because line classification into "left" and "right" is based on the slope of detected line segments. In a curve the slope of both left and right line segments is the same - and therefore the algorithm doesn't work well.

b) A contrasting object within the masked region (like a white car in front, arrow/writing on the road or other objects on the sides of the road) will (probably) mess up the output - as it might be classified as a line segment.

c) My intrapolation and extrapolation method is rather simplistic, just based on drawing a straight line between the "highest" and "lowest" ends of the detected line segments. It doesn't "weight" the segments or ignore "outliers" (data points not in line with the "line trend").

## 3. Suggest possible improvements to your pipeline

Following up on the shortcomings above, I would see the following possible improvements:

a) Create bendy lines - probably one way to do this would be to intrapolate between each and every detected line segment - so if they form a curve the output would also be a curve.

b) Identify areas / "corridors" with highest intensity of detected line segments and ignore data points that don't fit into these "corridors" - to avoid that random contrasting objects skew the output.

c) One other way to do this could be to introduce some sort of "memory" between the images in the video stream - and define by how much the position of the lines in the subsequent image can be different than in the previous image (+/- fluctuation corridor) - and ignore data points / detected  "line segments" which are outside of the fluctuation corridor. My assumption here would be that - taking into account video stream's frames-per-second speed and the car's speed - the lines' position should not fluctuate much between the subsequent images.

d) Probably there are many more ways to improve it :-)